


# Welcome to React Native Workshop

Learn to build mobile application using react native.

Speakers 

Jerin John K

Shailesh Yadav

Prerna Nagar

# What is React Native?

React Native is an open source framework for building Android and iOS applications using React and the app platform's native capabilities.

With React Native, you use **JavaScript** to access your platform's APIs as well as to describe the appearance and behavior of your UI using React components: bundles of reusable, nestable code.



# Get Started Without a Framework

<https://reactnative.dev/docs/getting-started-without-a-framework>

```
npx @react-native-community/cli@latest init rnWorkshop
```

```
Need to install the following packages:
```

```
@react-native-community/cli@13.6.6
```

```
Ok to proceed? (y) y
```

```
✓ Installing dependencies
```

```
✓ Do you want to install CocoaPods now? Only needed if you run your project in Xcode directly ... (y) y
```

```
Run instructions for iOS:
```

- `cd "/Users/jerin/rn/demo/rnWorkshop"`
- `npx react-native run-android`

# Folder structure and keywords

## JSX

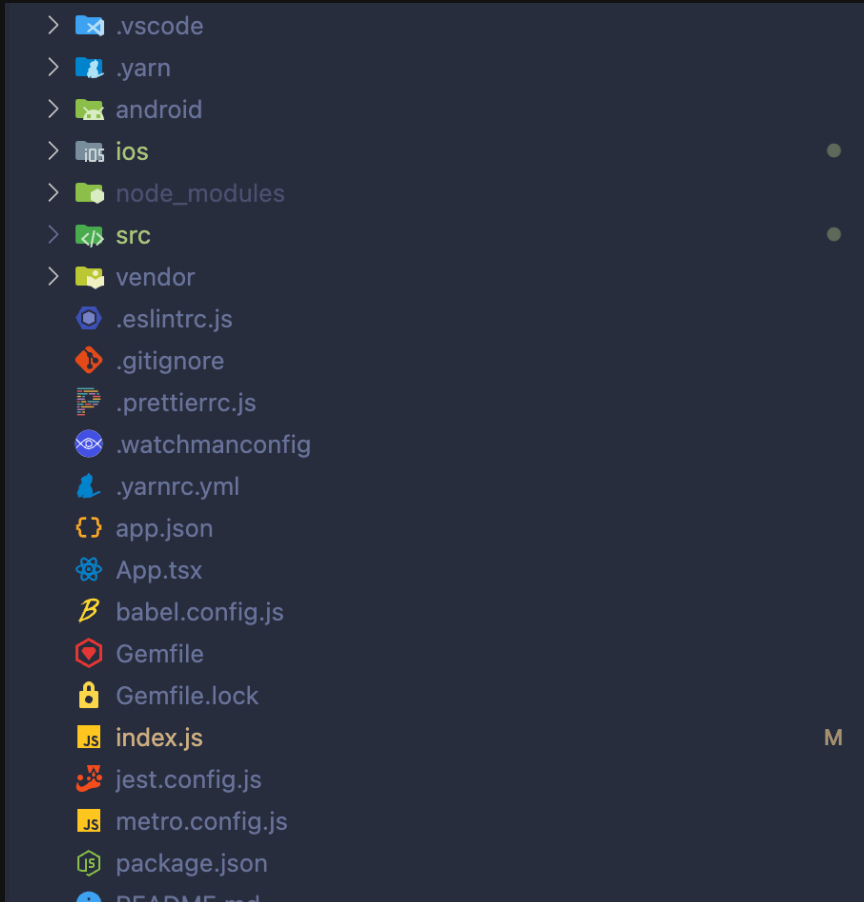
React and React Native use **JSX**, a syntax that lets you write elements inside JavaScript like so:

```
`<Text>Hello, I am your cat!</Text>`.
```

## Props

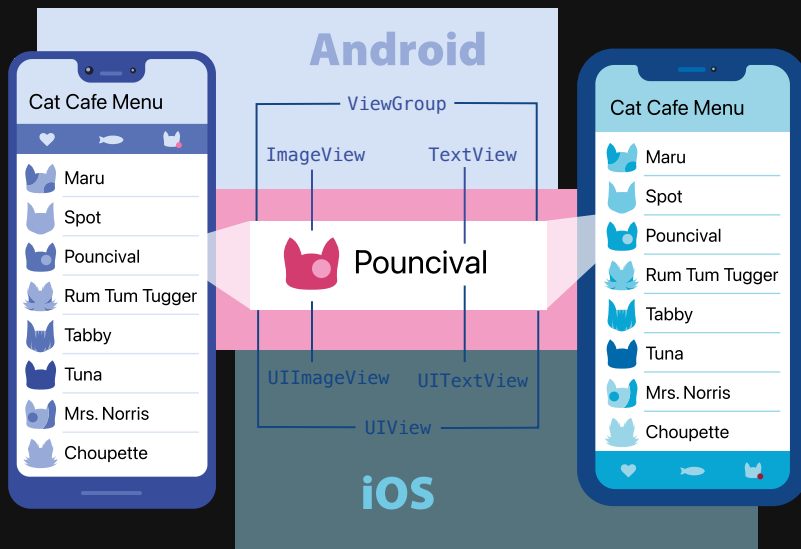
**Props** is short for “properties”. Props let you customize React components.

## State



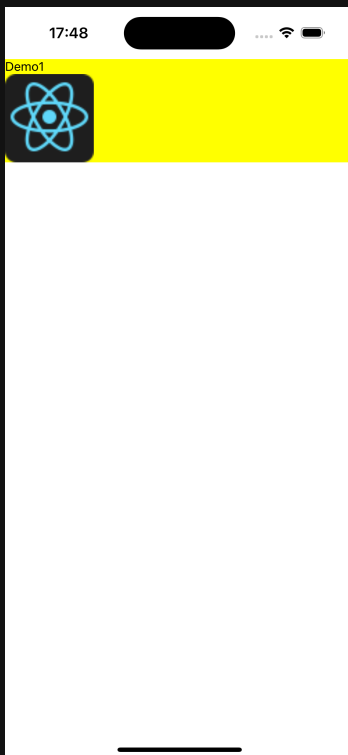
# Views and mobile development

In Android and iOS development, a view is the basic building block of UI: a small rectangular element on the screen which can be used to display text, images, or respond to user input. Even the smallest visual elements of an app, like a line of text or a button, are kinds of views. Some kinds of views can contain other views. It's views all the way down!



# Core Components

React Native UI Component	Android View	iOS View	Web Analog	Description
<code>&lt;View&gt;</code>	<code>&lt;ViewGroup&gt;</code>	<code>&lt;UIView&gt;</code>	A non-scrolling <code>&lt;div&gt;</code>	A container that supports layout with flexbox, style, some touch handling, and accessibility controls
<code>&lt;Text&gt;</code>	<code>&lt;TextView&gt;</code>	<code>&lt;UITextView&gt;</code>	<code>&lt;p&gt;</code>	Displays, styles, and nests strings of text and even handles touch events
<code>&lt;Image&gt;</code>	<code>&lt;ImageView&gt;</code>	<code>&lt;UIImageView&gt;</code>	<code>&lt;img&gt;</code>	Displays different types of images
<code>&lt;ScrollView&gt;</code>	<code>&lt;ScrollView&gt;</code>	<code>&lt;UIScrollView&gt;</code>	<code>&lt;div&gt;</code>	A generic scrolling container that can contain multiple components and views
<code>&lt;TextInput&gt;</code>	<code>&lt;EditText&gt;</code>	<code>&lt;UITextField&gt;</code>	<code>&lt;input type="text"&gt;</code>	Allows the user to enter text



```
1 // step 5 Fix:- Inline styling
2 import React from 'react';
3 import {Text, Image, View, SafeAreaView, StyleSheet} from 'react-native';
4
5 function Demo1() {
6   return (
7     <SafeAreaView>
8       <View style={styles.container}>
9         <Text>Demo1</Text>
10        <Image
11          source={{
12            uri: 'https://reactnative.dev/img/tiny_logo.png',
13          }}
14          style={styles.image}
15        />
16      </View>
17    </SafeAreaView>
18  );
19 }
20
21 const styles = StyleSheet.create({
22   container: {
23     backgroundColor: '#FF0',
24   },
25   image: {
26     width: 100,
27     height: 100,
```

# Basic Components



# View

`View` is a container that supports layout with flexbox, style, some touch handling, and accessibility controls.

```
import React from 'react';
import {View, Text} from 'react-native';

const ViewDemo = () => {
  return (
    <View
      style={{
        flexDirection: 'row',
        height: 100,
      }}>
      <View style={{backgroundColor: 'blue', flex: 0.3}} />
      <View style={{backgroundColor: 'red', flex: 0.5}} />
      <Text>Hello World!</Text>
    </View>
  );
};

export default ViewDemo;
```

`View` is designed to be nested inside other views and can have 0 to many children of any type.

# Text

A React component for displaying text. `Text` supports nesting, styling, and touch handling.

```
import React from 'react';
import {Text, StyleSheet} from 'react-native';

const TextDemo = () => {
  const titleText = 'Hello World';

  return <Text style={styles.titleText}>{titleText}</Text>;
};

const styles = StyleSheet.create({
  titleText: {
    fontSize: 20,
    fontWeight: 'bold',
  },
});

export default TextDemo;
```

# Image

A React component for displaying different types of images, including network images, static resources, temporary local images, and images from local disk, such as the camera roll.

```
import React from 'react';
import {Image, StyleSheet} from 'react-native';

const ImageDemo = () => (
  <Image
    style={styles.tinyLogo}
    source={{
      uri: 'https://reactnative.dev/img/tiny_logo.png',
    }}
  />
);

const styles = StyleSheet.create({
  tinyLogo: {
    width: 50,
    height: 50,
  },
});

export default ImageDemo;
```

# TextInput

Component for inputting text via a keyboard. Props provide configurability for several features, such as auto-correction, auto-capitalization, placeholder text, and different keyboard types, such as a numeric keypad.

You can also subscribe to the `onChangeText`, `onSubmitEditing` and `onFocus` events to read the user input.

```
export const TextInputDemo = () => {
  const [text, onChangeText] = React.useState('Placeholder Text');

  return (
    <TextInput style={styles.input} onChangeText={onChangeText} value={text} />
  );
};

const styles = StyleSheet.create({
  input: {
    height: 40,
    margin: 12,
    borderWidth: 1,
    padding: 10,
  },
});
```

# ScrollView

`ScrollViews` must have a bounded height in order to work, since they contain unbounded-height children into a bounded container

```
const ScrollViewDemo = () => {
  return (
    <ScrollView
      style={styles.container}
      contentContainerStyle={styles.scrollContent}>
      <Text>Lorem ipsum dolor sit</Text>
    </ScrollView>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
  },
  scrollContent: {
    backgroundColor: 'pink',
    marginHorizontal: 20,
  }
});

export default ScrollViewDemo;
```

# StyleSheet

A StyleSheet is an abstraction similar to CSS StyleSheets

- By moving styles away from the render function, you're making the code easier to understand.
- Naming the styles is a good way to add meaning to the low level components in the render function.

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    padding: 24,
    backgroundColor: '#eaeaea',
  },
  title: {
    marginTop: 16,
    paddingVertical: 8,
    borderWidth: 4,
    borderColor: '#20232a',
    borderRadius: 6,
    color: '#20232a',
    textAlign: 'center',
    fontSize: 30,
    fontWeight: 'bold',
  },
});
```

# User Interfaces

# Button

A basic button component that should render nicely on any platform. Supports a minimal level of customization.

```
<Button
  onPress={onPressLearnMore}
  title="Learn More"
  color="#841584"
  accessibilityLabel="Learn more about this purple button"
/>
```

You can build your own button using **Pressable**

```
<Pressable onPress={onPressFunction}>
  <Text>I'm pressable!</Text>
</Pressable>
```



# Switch

Renders a boolean input.

This is a controlled component that requires an `onVaLueChange` callback that updates the `value` prop in order for the component to reflect user actions.

```
const SwitchDemo = () => {
  const [isEnabled, setIsEnabled] = useState(false);
  const toggleSwitch = () => setIsEnabled(previousState => !previousState);

  return (
    <Switch
      trackColor={{false: '#767577', true: '#81b0ff'}}
      thumbColor={isEnabled ? '#f5dd4b' : '#f4f3f4'}
      ios_backgroundColor="#3e3e3e"
      onVaLueChange={toggleSwitch}
      value={isEnabled}
    />
  );
};

export default SwitchDemo;
```

# FlatList

A performant interface for rendering basic, flat lists, supporting the most handy features:

- Fully cross-platform.
- Optional horizontal mode.
- Configurable viewability callbacks.
- Header support.
- Footer support.
- Separator support.
- Pull to Refresh.
- Scroll loading.
- ScrollToIndex support.
- Multiple column support.

Shopify's FlashList is a fast & performant list

# Layout with Flexbox

# What is flexbox

A component can specify the layout of its children using the Flexbox algorithm. Flexbox is designed to provide a consistent layout on different screen sizes.

We're primarily going to use a combination of `flexDirection`, `alignItems`, and `justifyContent` to achieve the right layout.

# Let's Flex 💪

`flex`` will define how your items are going to `fill`` over the available space along your main axis. Space will be divided according to each element's flex property.

```
const FlexDemo = () => {  
  return (  
    <View  
      style={{ flex: 1, padding: 20, flexDirection: "column", }}> // change  
      <View style={{ flex: 1, backgroundColor: "red" }} />  
      <View style={{ flex: 2, backgroundColor: "orange" }} />  
      <View style={{ flex: 3, backgroundColor: "green" }} />  
    </View>  
  );  
};  
  
export default FlexDemo;
```

# Flex Direction

`flexDirection`` controls the direction in which the children of a node are laid out.

This is also referred to as the main axis.

- `column`` (default value) Aligns children from top to bottom.
- `row`` Aligns children from left to right.
- `column-reverse`` Aligns children from bottom to top.
- `row-reverse`` Aligns children from right to left.

Note: In web `flexDirection`` defaults to column instead of row

# Justify Content ≡

`justifyContent`` describes how to align children within the **main axis** of their container.

- `flex-start`` (default value)
- `flex-end``
- `center``
- `space-between``
- `space-around``
- `space-evenly``

Note: Compared to `space-between`` using `space-around`` will result in space being distributed to the beginning of the first child and end of the last child.

# Align Items 🤔

`alignItems`` describes how to align children along the **cross axis** of their container.

- `stretch`` (default value)
- `flex-start``
- `flex-end``
- `center``
- `baseline``

Info: For `stretch`` to have an effect, children must not have a fixed dimension along the secondary axis



## Align Self 🤔

`alignSelf`` has the same options and effect as `alignItems` but instead of affecting the children within a container, you can apply this property to a single child to change its alignment within its parent. It overrides any option set by `alignItems`.

## Flex Wrap 📦

The `flexWrap`` property is set on containers and it controls what happens when children overflow the size of the container along the main axis.

# Absolute & Relative Layout 🤪

The position type of an element defines how it is positioned within its parent.

- ``relative`` (default value) element is positioned according to the normal flow of the layout.
- ``absolute`` element doesn't take part in the normal layout flow. The position is determined based on the top, right, bottom, and left values.

```
<View
  style={{
    width: 50,
    height: 50,
    top: 50,
    left: 50,
    position: 'absolute',
    backgroundColor: 'skyblue',
  }}
/>
```

Hooks

React Hooks provide functional components with the ability to use states and manage side effects. They were first introduced in React 16.8. They provide a cleaner and more concise way to handle state and side effects in React applications.

Hook Rules There are 3 rules for hooks:

- Hooks can only be called inside react function components.
- Hooks can only be called at the top level of a component.
- Hooks cannot be conditional

# useState

`useState` is a React Hook that lets you add a state variable to your component.

```
const [state, setState] = useState(initialState)
```

Call `useState` at the top level of your component to declare one or more state variables.

```
import { useState } from 'react';

function MyComponent() {
  const [count, setCount] = useState(0);
  // ...
}
```

The convention is to name state variables like `[something, setSomething]` using array destructuring.

`useState` returns an array with exactly two items

- The current state of this state variable, initially set to the initial state you provided.
- The set function that lets you change it to any other value in response to interaction.

# useEffect

It is called every time any state if the dependency array is modified or updated

```
useEffect(setup, dependencies?)
```

- `setup` : The function with your Effect's logic. Your setup function may also optionally return a cleanup function.
- optional `dependencies` : List of Reactive values include props, state, and all the variables and functions declared directly inside your component body.

```
useEffect(() => {  
  const connection = createConnection(serverUrl, roomId);  
  connection.connect();  
  return () => {  
    connection.disconnect();  
  };  
}, [serverUrl, roomId]);
```

# UseEffect example

On each button press we are incrementing the counter and showing its updated value.

```
import React, {useState, useEffect} from 'react';
import {Button} from 'react-native';

export default function StatesDemo() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    console.log(count);
  }, [count]);

  return (
    <Button
      title={`Increment ${count}`}
      onPress={() => {
        // setCount(count + 1);
        setCount(prevState => prevState + 1); // asynchronous
      }}
    />
  );
}
```

# useRef

useRef is a React Hook that lets you reference a value that's not needed for rendering.

```
const ref = useRef(initialValue)
```

# useCallback

useCallback is a React Hook that lets you cache a function definition between re-renders.

```
const cachedFn = useCallback(fn, dependencies)
```

# useMemo

useMemo is a React Hook that lets you cache the result of a calculation between re-renders.

```
const cachedValue = useMemo(() => {  
  // .. large calculation  
  return calculatedValue;  
}, dependencies)
```



# Networking

React Native provides the Fetch API for your networking needs. Networking is an inherently asynchronous operation.

```
const getMoviesFromApi = () => {  
  return fetch('https://reactnative.dev/movies.json')  
    .then(response => response.json())  
    .then(json => {  
      return json.movies;  
    })  
    .catch(error => {  
      console.error(error);  
    });  
};
```

Third party libraries such as axios can be added and used instead.

# Navigating Between Screens

React Navigation is used for managing the presentation of, and transition between, multiple screens which is typically handled a navigator.

```
npm install @react-navigation/native @react-navigation/native-stack
npm install react-native-screens react-native-safe-area-context
```

```
import {NavigationContainer} from '@react-navigation/native';

const App = () => {
  return (
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen
          name="Home"
          component={HomeScreen}
          options={{title: 'Welcome'}}
        />
        <Stack.Screen name="Profile" component={ProfileScreen} />
      </Stack.Navigator>
    </NavigationContainer>
  );
};
```

- You can set options such as the screen title for each screen in the `options` prop of `Stack.Screen`.
- Each screen takes a `component` prop that is a React component.
- Those components receive a prop called `navigation` which has various methods to link to other screens. For example, you can use `navigation.navigate` to go to the `Profile` screen:

```
const HomeScreen = ({navigation}) => {  
  return (  
    <Button  
      title="Go to Jane's profile"  
      onPress={() =>  
        navigation.navigate('Profile', {name: 'Jane'})  
      }  
    />  
  );  
};  
  
const ProfileScreen = ({navigation, route}) => {  
  return <Text>This is {route.params.name}'s profile</Text>;  
};
```

# Animations

React Native provides two complementary animation systems: `Animated`` for granular and interactive control of specific values, and `LayoutAnimation`` for animated global layout transactions.

## Animated API

`Animated`` focuses on declarative relationships between inputs and outputs, with configurable transforms in between, and `start` / stop`` methods to control time-based animation execution.

## LayoutAnimation API

`LayoutAnimation`` allows you to globally configure `create`` and `update`` animations that will be used for all views in the next render/layout cycle.

With react-native-reanimated, you can easily create smooth animations and interactions that run on the UI thread.

# Storage and State Managers

## Async Storage

To store string data locally on device.

```
npm install @react-native-async-storage/async-storage
```

react-native-mmkv-storage is a lightweight and faster alternative

## Redux toolkit

This includes store setup, creating reducers and writing immutable update logic, and even creating entire ``slices`` of state at once.

```
npm install @reduxjs/toolkit  
npm install redux
```

Zustand and Jotai are a great alternative to redux to reduce boilerplate code.

# Debugging

React Native provides an in-app developer menu which offers several debugging options. You can access the Dev Menu by shaking your device or via keyboard shortcuts:

- iOS Simulator: `Cmd ⌘ + D` (or Device > Shake)
- Android emulators: `Cmd ⌘ + M` (macOS) or `Ctrl + M` (Windows and Linux)

Alternatively for Android devices and emulators, you can run ``adb shell input keyevent 82`` in your terminal.

**Reactotron** can monitor application's state, network requests, and performance metrics.

```
npx react-devtools
```

```
npx react-native start --experimental-debugger
```

or press `J`

# Recommendation

- *Avoid UI re-renders*
  - UI flickering
  - Frames dropping
- Make use of ``Typescript`` for writing code and ``Storybook`` for documentations.
- Use dedicated Components like `flashList`, `fast-image` etc
- Always animate at 60 FPS
- Create pure components while dividing your components into presentation and container components while keeping your UI and business logic separate.

# Expo

Expo is a production-grade React Native **Framework**. Expo provides developer tooling that makes developing apps easier, such as file-based routing, a standard library of native modules, and much more.

Expo's Framework is free and open source, with an active community on GitHub and Discord. The Expo team works in close collaboration with the React Native team at Meta to bring the latest React Native features to the Expo SDK.

The team at Expo also provides Expo Application Services (EAS), an optional set of services that complements Expo, the Framework, in each step of the development process.



# Native Modules

The NativeModule system exposes instances of Java/Objective-C/C++ (native) classes to JavaScript (JS) as JS objects, thereby allowing you to execute arbitrary native code from within JS.

- Creating a local library that can be imported in your React Native application.
- Directly within your React Native application's iOS/Android projects
- As a NPM package that can be installed as a dependency by your/other React Native applications.

Thank You